ＣＯＢＯＬによる２０００年問題

The Year 2000 Problem through COBOL

武　田　嘉　孝

１．はじめに

西暦「２０００年」で発生するコンピュータの誤作動に対して、産業界では情報システムの改修対策を緊急に講じなければならなくなった。この問題が発生するのも古いコンピュータの記憶容量を節約するため、西暦の上二桁、１９を省略し、下二桁の００を 1900 年と読ませるようにプログラムを組んでいたことに起因する。2000 年の下二桁を、1900 年と混同するため、誤作動を起こしてしまうことになる。すでにこうした問題を想定して、国際標準化機構（ＩＳＯ）が 1989 年から四桁表示を標準規格に採用している。また、日本工業規格（ＪＩＳ）でも 1992 年から四桁の標準化を行っている。しかしながら、企業が情報システムを更新するときにおいても、データを下二桁で表示を続けてきたケースが多く見受けられた。

問題解決の対策として、プログラムの修正および運用テストの時間と、莫大な費用が必要となる。当面はソフトを修正し、外部システムとの接続テストを実施しなければならない。自社のプログラム修正と運用テストにとどまらず、ネットワーク化した取引先との運用テストをしてみて、問題の発生が見られないかをチェックしなければならない。十分に対応できない場合、被害やトラブルを最小限にするために、必要なデータ入力だけにして、外部との接続手続きを外すなどの具体的運用対策を講じるなどによって、ネットワーク・トラブルを防ぐことも考慮しなければならない。

さて、プログラムの修正ならびに運用テスト、さらに外部取引先とのネットワーク接続による、「2000 年問題」対策に対処できなければ、金融機関の「資金決済システム」およびクレジット・カードの情報管理が、間違った請求額や期限切れの情報を流したりする。またネットワーク上のどこか１ヶ所でも問題が生じれば、間違ったデータを出力し、連鎖的に被害が発生する恐れもある。たとえば、電子機器を制御するために用いられているマイコンの制御が行えなくなり、曜日などが正確に管理できないため、平日の操業日に工場のプラント操業に支障をもたらしたり、銀行のキャッシュカードがＡＴＭの誤作動により、

休日扱いによる付加料金が引き落とされたり、電話料金の休日割引料金が正しく計算されないで、顧客に平日料金を請求するなどコンピュータ誤作動によるネットワーク・トラブルが続出し、国民の社会生活において大混乱を発生させることになる。

本稿では、金融機関や官公庁などで用いられている事務用共通言語、ＣＯＢＯＬプログラムの「２０００年問題対策」に焦点を絞り、具体的なプログラム修正の在り方を検討することにしたい。

２．ＣＯＢＯＬの現状と歴史

1980 年代後半に、ダウンサイジングが叫ばれ、コンピュータがメインフレーム中心からワークステーション（WS）、パソコン（PC）の普及への変化がみられた。その当時は、ＷＳ/ PC 用の COBOL が少なく、Ｃ言語に取って代わられる時期が存在した。その頃、巷では、これからは「Visual Basic や Java やＣ／Ｃ++ 」という神話すら生れたほどである。これらの言語は、ｺﾝﾋﾟｭｰﾀ・ｸﾞﾗﾌｨｯｸｽやＧＵＩ（Graphical User Interface）などの派手な分野でもてはやされているきらいがある。これに対して、目立たないとはいえ、COBOL プログラムは、現実に企業・官庁などの業務システムの基幹システムとして活用されている。具体的には、銀行オンラインシステム、電力会社の料金システム、各種保険システム、給与明細発行システム、水道・ガス、戸籍等のシステム、税務関係電算システム、財務会計システムにも用いられ、今では社会的インフラとしての地位を確保するまでに至っていると言っても過言ではない。

1990 年代に入り、ＷＳ/ PC のハードウエアや OS において飛躍的な発展がみられた。高性能 CPU の出現により、処理能力の高速化と信頼性の増大、OS についても、３２ビット OS（Windows NT と Windows 95 ）の登場によって信頼性が増してきたことである。これによって、WS / PC 上でも基幹業務システムを構築できるだけのミドルウェア（システム運用管理、データベース管理、オンライントランザクションモニタなど）が充実して、ＷＳ/ PC 上でも業務システムが組めるようになった。このような環境的な整備が図られたために、1993 年頃から COBOL については、Windows 対応 COBOL が発表され、最新の PC 用の COBOL は、NT COBOL と称して、各社の製品が市場に出回っている。COBOL が、1960 年に誕生してから、2000 年発表予定の ISO COBOL 2000（仮称）に至るまでの、発展の歴史を振り返ってみよう。

ＣＯＢＯＬの歴史

①CODASYL (The COnference on DAta SYstems Languages) ﾃﾞｰﾀｼｽﾃﾑｽﾞ言語協議会

CODASYL コボルは、全てのｺﾎﾞﾙの原典であり、規格をしないのが特徴である。

1960年　COBOL－６０発表
1961年　COBOL－６１（60年の改訂版で、実質的な普及版）
1963年　拡張COBOL－６１
1965年　COBOL 1965（大記憶）
1968年　COBOL 1968（ﾌﾟﾛｸﾞﾗﾑ間連絡）
1969年　COBOL 1969（通信機能）
1970年　COBOL 1970（ﾃﾞﾊﾞｯｸﾞ 併合）
1973年　COBOL 1973
1976年　COBOL 1976（ﾃﾞｰﾀﾍﾞｰｽ）
1978年　COBOL 1978（整構造）
1981年　COBOL 1981
1984年　COBOL 1984
1989年　拡張COBOL

② ISO（ International Organization for Standardization ）国際標準化機構

ISOコボル
1972年　ISO COBOL
1978年　ISO COBOL（第１回改訂）
1985年　ISO COBOL
2000年　ISO COBOL 2000（仮称）予定

③ ANSI（ American National Standards Institute ）米国規格協会

ANSコボル
1968年　ANS COBOL（COBOL 1965に準拠）
1974年　ANS COBOL（第１回改訂、COBOL 1970に準拠）
1985年　ANS COBOL（第２回改訂）

④ JIS （ Japanese Industrial Standards ）日本工業規格

JISコボル
1972年　JIS COBOL（第１次規格誕生、1972 ISO COBOLに準拠）
1980年　JIS COBOL（第２次規格、1978 ISO COBOLに準拠）
1988年　JIS COBOL（第３次規格、1985 ISO COBOLに準拠）

⑤ MS-DOS用COBOL

1982年　日本語LEVEL II COBOL（1974 ANS COBOLに準拠、ﾏｲｸﾛﾌｫｰｶｽ社）
1988年　日本語COBOL/２（L/II COBOLの改訂版、1985 ANS COBOLに準拠）

この他、（NEC）PC COBOL
（富士通）PowerCOBOL

⑥ **Windows 対応 COBOL → NT COBOL**

1993 年　MF-COBOL V3.1J　（1985 ANS、ISO COBOL に準拠、ﾏｲｸﾛﾌｫｰｶｽ社）
1996 年　MF-COBOL V4.0J、1998 年　MF-COBOL V5.0J
1999 年　Micro Focus Net Express 3.0J
この他、（NEC）　Open COBOL Factory
（富士通）Power COBOL 97 V5.0
（日立）　COBOL 85 V5.0
（米国ｱｷｭｺﾎﾞﾙ）ACU COBOL-GT

3．2000 年問題の意味

2000 年問題は、「Ｙ２Ｋ」とも略称され、20 世紀末の重要な情報システム問題である。その問題がどういうものであるかを定義し、どうしてそれが発生するのか、そしてそれに対処する方法を具体的に検討してみることにしよう。

2000 年問題は、また 2 桁日付問題または 1000 年紀問題として知られており、大抵のコンピュータやアプリケーションが、1900 年と 2000 年との区別ができない問題と言うことができる。コンピュータは、実際には今世紀に導入され、1980 年代に普及したばかりである。ここ最近、ハードウェアやソフトウェアの販売業者が 2000 年問題に柔軟に対処しうるように活動を展開している。そもそもこのような問題が発生するのは、1996 年以前に製造された大抵のコンピュータ、オペレーティング・システム、ソフトウェア・パッケージが、次の世紀の変わり目に日付を処理することができないからである。

すなわち、1960 年代から 1980 年代にかけて、多くのプログラムなどを含むシステムに、日付が必要なときはいつでも年の最後の２桁だけを用いたファイルを開発したためである。長い間このような実務は、何の問題もなく作用した。したがってその当時としては、特殊な問題を引き起こすような事態になるとは予想されてはいなかった。そうであるからといってなぜシステム開発者が、前もって 4 桁年を標準的な実務として用いなかったのであろうか。

それに対する１つ目の答えは、その当時に使用していたハードウェアに原因がある。メインフレーム（大型コンピュータ）がその当時は主要なコンピュータであり、それが大抵のシステムを実行していた。これらはマシーン上で、ディスク記憶容量、主記憶容量に膨大な費用がかかった。したがってプログラマの立場からは、記憶容量を経済的に使用する必要性に迫られていた。そのため年についても、２桁の不必要な桁を削除することが経費削減のための短絡的な解決策となり、そのような意図の下でなされた意思決定は、莫大な

資金を少しでも節約するために行われた恣意的なものであったと言える。

２つ目の答えは、COBOL コンパイラの中にある。１つ目と同じロジックを用いることで、「ACCEPT ― ― ― FROM DATE.」コマンドは、YYMMDD 形式でシステム日付を取り出した。この日付形式は、もっと新しい規格の下ではうまく処理されない。4 桁年を維持するためにプログラマは、特別の桁は作れるところはできる限り年領域に計画的に確保するように加算しなければならない。プログラマの見方からすると、この当時一番楽な方法は、まさに２桁年を用いることであったと言えよう。

３つ目の答えは、大抵のシステム開発者は、これらのプログラムがそうした問題に対してこれほど大きく問題化するほど十分長くまで存在しつづけるものとは考えていなかったためである。彼らの抱く期待は、プログラムは 2000 年になる前に書き直されるであろうし、その結果新しいシステムが、その問題を処理するであろう、ということであった。しかしながら、それから 20～30 年後、これらの「遺産」システムは、今なお稼働しているため、そのつけがいま大きな問題となって、全世界に不安と緊張をもたらしている。

2000 年問題は、世界中のコンピュータ・システムの約 80%に影響を与え、解決をするのに、3000～6000 億ドルもの経費がかかると言われている。すでにその作業のいくつかは終了している。けれども、たとえば、クレジットカード会社のような経営活動にあるように、将来の日付を計画する必要のある会社は、もう数年も前からこの問題に係りきりになっている。こうした会社は、幸先よくスタートを切り、もうその作業を終了しているところもある。金融機関を中心に全部ではないにしても、今までに換算作業が大抵は終了しているとみられる。

この問題の大部分は、4 桁日付をコンピュータのハードウェアやソフトウェアが、上手く処理できないことである。大抵、実業界では、２桁年をもつ日付でアプリケーションを利用する。たとえば、1995 年は、95 という数字でコンピュータやコンピュータ・プログラムで示される。２桁年は、たとえば、99 から 00 に当然に変化していく。しかしながら、コンピュータは、1900 年と 2000 年の区別をすることができていないのである。したがって、この現象から、2000 年問題によって世界中の全てのコンピュータが、2000 年１月１日から正常に操作しなくなる事態が発生することが心配されている。なぜコンピュータは、正常に操作できなくなるのであろうか。その理由は、日付を処理するプログラムが、2000 年を操作できるようにプログラム化されていないからである。

日付は、典型的に mmddyy として格納され、そしてそこでの世紀数字はやめになり、1900 年が仮定される。2000 年がやってくるので、日付の年は、00 となるし、コンピュータ・プログラムは、それが 1900 年であることを仮定するか、一斉に止まってしまうかのどちらかになるであろう。その理由は、プログラムが 00 年を使用できないためである。

現実の 2000 年問題は、日付が処理されている全てのプログラムに対して、それに対する解決法を見つけ、それから 2000 年１月１日までに全てのプログラムを更新し、検査することを試みている。

つぎに示すプログラムは、現在の日付から誕生年を差し引くことによって、人の年を計算するものである。現在の日付は、DATE リクエストを用いているオペレーティング・システムからの日付を要求することによるプログラムから求められる。これは全て、システムの日付が 2 桁か 4 桁か、そしてオペレーティング・システムが 2000 年に柔軟であるかどうかに依存している。

```
001010 IDENTIFICATION        DIVISION.
001020 PROGRAM-ID.           PROG1.
001030 ENVIRONMENT           DIVISION.
001040 INPUT-OUTPUT          SECTION.
001050 FILE-CONTROL.
001060 DATA                  DIVISION.
001070 WORKING-STORAGE       SECTION.
001080 01  BIRTH-DATE.
001090     05   BIRTH-YR     PIC 9(2).
001100     05   BIRTH-MO     PIC 9(2).
001110     05   BIRTH-DAY    PIC 9(2).
001120 01  CUR-DATE.
001130     05   CUR-YR       PIC 9(2).
001140     05   CUR-MO       PIC 9(2).
001150     05   CUR-DAY      PIC 9(2).
001160 01  AGE               PIC S9(3).
001170 PROCEDURE              DIVISION.
001180     MOVE  80  TO  BIRTH-YR.
001190     ACCEPT  CUR-DATE  FROM  DATE.
001200     COMPUTE  AGE  =  CUR-YR  -  BIRTH-YR.
001210     DISPLAY  AGE.
001220     STOP     RUN.
```

さて、さらに日付に関する 3 つのプログラムをみてみよう。COBOL プログラムでは、予約語、DATE を用いることによって、オペレーティング・システムへのリクエストによって、現在の日付を判断する。DATE は、システム日付としてオペレーティング・システムに登録し、それからアプリケーションにその価値を返す。そのリクエストは、ある変数を初めに作ることによって普通は行われる。その変数は、アプリケーションの中にその日付を格納するために用いられる。その変数は、PIC　X(6) として定義される。

つぎに、ACCEPT コマンドは、オペレーティング・システムから日付のリクエストを行うために用いられる。それから日付変数に引取価値を割り当てる。これはつぎのプログラムの例示からも明らかとなる。また FROM 句は、文の中で用いられる。FROM 句は、ACCEPT 命令がデータ変数に割り当てるためのデータをどこに見つけるべきかを確認するものである。

```
001010 IDENTIFICATION        DIVISION.
001020 PROGRAM-ID.           PROG2.
001030 ENVIRONMENT           DIVISION.
001040 INPUT-OUTPUT          SECTION.
001050 FILE-CONTROL.
001060 DATA                  DIVISION.
001070 WORKING-STORAGE       SECTION.
001080 01  BIRTH-DATE.
001090     05   BIRTH-YR     PIC 9(2).
001100     05   BIRTH-MO     PIC 9(2).
001110     05   BIRTH-DAY    PIC 9(2).
001120 01  CUR-DATE.
001130     05   CUR-YR       PIC 9(2)  VALUE 00.
001140     05   CUR-MO       PIC 9(2).
001150     05   CUR-DAY      PIC 9(2).
001160 01  AGE               PIC S9(3).
001170 PROCEDURE              DIVISION.
001180     MOVE  80  TO  BIRTH-YR.
001190     ACCEPT  CUR-DATE  FROM  DATE.
001200     COMPUTE  AGE  =  CUR-YR  -  BIRTH-YR.
001210     DISPLAY  AGE.
001220     STOP     RUN.
```

```
001010 IDENTIFICATION        DIVISION.
001020 PROGRAM-ID.           PROG3.
001030 ENVIRONMENT           DIVISION.
001040 INPUT-OUTPUT          SECTION.
001050 FILE-CONTROL.
001060 DATA                  DIVISION.
```

```
001070 WORKING-STORAGE        SECTION.
001080 01  CUR-DATE.
001090     05   CUR-YR        PIC 9(2).
001100     05   CUR-MTH       PIC 9(2).
001110     05   CUR-DAY       PIC 9(2).
001120 PROCEDURE              DIVISION.
001130     ACCEPT  CUR-DATE  FROM  DATE.
001140     DISPLAY  "今日の日付は" CUR-DATE  "です。".
001150     STOP     RUN.
```

```
001010 IDENTIFICATION         DIVISION.
001020 PROGRAM-ID.            PROG4.
001030 ENVIRONMENT            DIVISION.
001040 INPUT-OUTPUT           SECTION.
001050 FILE-CONTROL.
001060 DATA                   DIVISION.
001070 WORKING-STORAGE        SECTION.
001080 01  CUR-DAY.
001090     05   CUR-YR        PIC 9(2).
001100     05   CUR-DAY-VAL   PIC 9(4).
001110 PROCEDURE              DIVISION.
001120     ACCEPT  CUR-DAY   FROM  DAY.
001130     DISPLAY "今日は" CUR-YR "年１月１日から" CUR-DAY-VAL "日目".
001140     STOP     RUN.
```

これらのプログラムは、システムから日付を取り出す方法であった。しかしながら用意されている日付年の桁数は２桁であった。したがってこれからは、プログラマは日付年の領域として４桁を組み込むように変更をしなければならない。そうした作業も、あまり多くの日付領域に関するところがない場合などでは簡単なものとなる。しかしながらプログラマとしては、年をあらわす領域の全てを見つけ変更し、さらにまた印刷形式の部分も訂正しなければならない。もっと大きいプログラムの場合、プログラマはいくつかの必要な変更を見落としてしまう可能性が大きい。プログラマにとっては、プログラムの訂正に加えて、ファイル定義の変更、データの変更が必要となる。そのようにあるプログラムは、そのデータを新しい形式に換算しなければならない。このプロセスが継続している間、誰

かが他の関係するプログラムの全てを新しい形式のデータを受け入れるように換算しなければならない。たとえそれらのプログラムが日付領域を直接に利用しないとしても、それらのことが重要である。要するに、それらの変更それ自体は、簡単ではあるが、その含みと大きさは圧倒的なものになるはずである。たとえばいくつかの会社では、2000年問題によって影響をうける何千ものプログラムを有している。多くの組織体では、関係するファイル、画面、報告書の全てに亘って検査をし、変更するために5000万行ものコードをもつかもしれない。

４．2000年問題の対処法

2000年問題の調整のための対処法については、これまでCOBOLアプリケーションが、オペレーティング・システムからどのように日付情報を把握するかについてプログラム例を用いて紹介した。この情報は、日付処理を実行するルーチンを見つけるには、プログラムのどこを見ればよいのかについて理解できるきっかけを提供するものである。

日付を調整するための１つの方法は、ある仮定に基づいてデータ・ルーチンを書き直すことである。たとえばその仮定とは、50から99までの年が1950年から1999年までであり、00から49までの年が、2000年から2049年までとするものである。この解決法は、2000年問題を解決するための「調整ウィンドウ方法」と呼ばれるものである。その具体例のプログラムは、つぎに示すPROG5.CBLである。これは、１つのファイルの１領域のような日付のデータが、通常読み込まれ、２つの文字日付を含むレコードに割り当てられる。そのファイルにおけるデータの価値や形式は、変更しない。「調整ウィンドウ」ルーチンは、日付の年文字を評価し、前述の仮定についての価値となるようにする。

```
001010 IDENTIFICATION          DIVISION.
001020 PROGRAM-ID.             PROG5.
001030 ENVIRONMENT             DIVISION.
001040 INPUT-OUTPUT            SECTION.
001050 FILE-CONTROL.
001060     SELECT  OLD-FILE  ASSIGN  TO  "A:MYFILE1.DAT".
001070     SELECT  NEW-FILE  ASSIGN  TO  "A:MYFILE2.DAT".
001080 DATA                    DIVISION.
001090 FILE                    SECTION.
001100 FD  OLD-FILE.
001110 01  OLD-DATE.
001120     05   OLD-DATE-YR  PIC 9(2).
001130     05   OLD-DATE-MTH PIC 9(2).
```

```
001140      05   OLD-DATE-DAY PIC 9(2).
001150      05   FILLER       PIC X(2).
001160 FD  NEW-FILE.
001170 01  NW-DATE.
001180      05   NW-DATE-CC   PIC 9(2).
001190      05   NW-DATE-YR   PIC 9(2).
001200      05   NW-DATE-MTH  PIC 9(2).
001210      05   NW-DATE-DAY  PIC 9(2).
001220 PROCEDURE              DIVISION.
001230 A1. OPEN   INPUT  OLD-FILE  OUTPUT     NEW-FILE.
001240 A2. READ   OLD-FILE  AT  END  GO  TO  A3.
001250      MOVE   OLD-DATE-YR  TO  NW-DATE-YR.
001260      MOVE   OLD-DATE-MTH TO  NW-DATE-MTH.
001270      MOVE   OLD-DATE-DAY TO  NW-DATE-DAY.
001280      IF  OLD-DATE-YR  IS  >  50  MOVE  19  TO  NW-DATE-CC
001290                ELSE    MOVE  20  TO  NW-DATE-CC
001300      END-IF.
001310      WRITE  NW-DATE  AFTER  1.
001320      DISPLAY  NW-DATE.
001330      GO  TO  A2.
001340 A3. CLOSE  OLD-FILE  NEW-FILE.
001350      STOP     RUN.
```

あらゆるアプリケーションにおける 2000 年バグを取り除くことは、手間のかかる作業である。しかし、2000 年 1 月 1 日の期限は延期できないので、大抵のプログラムは調整をしなければならない。その問題に取り掛かり、最後までやろうとする情報システム管理者は、永久的な解決法が見つかるまで、その問題を一時的に調整するための「一時凌ぎの尺度」を探すものである。今現在、一時凌ぎの尺度の趨勢が、日付年についての 1 つの仮定を作るルーチンであることは既に述べたところである。典型的に、50 から 99 までの年は 1900 年代に入るものとされ、00 から 49 までの年は 2000 年代に入るものと仮定された。このロジックは、日付を用いるあらゆるアプリケーションの中に組み込まれる。こうした調整法を「調整ウィンドウ・アプローチ」という呼び方をされることもある。

これに対して、毎年、年をスライドさせてその仮定を自動的に変更するためのアプローチを、「スライド式ウィンドウ」または「スライド式ウィンドウ・アプローチ」という。この調整法を具体的に示すならば、たとえば 2000 年においては、50 から 99 年は 1900

年代であり、00 から 49 年は 2000 年以上であると仮定される。2001 年においては、51 から 99 年は 1900 年代であり、00 から 50 年は 2000 年に入ると仮定される。このシフト・パターンが毎年継続されるために、「スライディング」という名前が付けられたものである。

この場合の具体的なプログラム例は、つぎに示す、PROG5A.CBL である。

```
001010 IDENTIFICATION          DIVISION.
001020 PROGRAM-ID.             PROG5A.
001030 ENVIRONMENT             DIVISION.
001040 INPUT-OUTPUT            SECTION.
001050 FILE-CONTROL.
001060     SELECT  OLD-FILE  ASSIGN  TO  "A:MYFILE1.DAT".
001070     SELECT  NEW-FILE  ASSIGN  TO  "A:MYFILE2A.DAT".
001080 DATA                    DIVISION.
001090 FILE                    SECTION.
001100 FD  OLD-FILE.
001110 01  OLD-DATE.
001120     05   OLD-DATE-YR  PIC 9(2).
001130     05   OLD-DATE-MTH PIC 9(2).
001140     05   OLD-DATE-DAY PIC 9(2).
001150     05   FILLER       PIC X(2).
001160 FD  NEW-FILE.
001170 01  NW-DATE.
001180     05   NW-DATE-CC   PIC 9(2).
001190     05   NW-DATE-YR   PIC 9(2).
001200     05   NW-DATE-MTH  PIC 9(2).
001210     05   NW-DATE-DAY  PIC 9(2).
001220 PROCEDURE               DIVISION.
001230 A1. OPEN   INPUT  OLD-FILE  OUTPUT     NEW-FILE.
001240 A2. READ   OLD-FILE  AT  END  GO  TO  A3.
001250     MOVE   OLD-DATE-YR  TO  NW-DATE-YR.
001260     MOVE   OLD-DATE-MTH TO  NW-DATE-MTH.
001270     MOVE   OLD-DATE-DAY TO  NW-DATE-DAY.
001280     IF  OLD-DATE-YR  IS  >  51  MOVE  19  TO  NW-DATE-CC
001290                 ELSE    MOVE  20  TO  NW-DATE-CC
001300     END-IF.
```

```
001310      WRITE  NW-DATE  AFTER  1.
001320      DISPLAY  NW-DATE.
001330      GO  TO  A2.
001340 A3.  CLOSE  OLD-FILE  NEW-FILE.
001350      STOP     RUN.
```

つぎの問題は、曜日調整の方法についてである。いくらかのCOBOLアプリケーションは、曜日を参照する。もしそのアプリケーションが、2000年に柔軟でなく、1900年代の日付を仮定するならば問題が発生する。1900年1月1日は月曜日である。しかしながら、2000年1月1日は土曜日である。この違いは一般的に、曜日問題と呼ばれる。その年が1900年であることを仮定するアプリケーションは、不正確に曜日を計算するであろう。この問題は曜日を利用する全てのアプリケーションにとって、関わりのあるものである。たとえば、多少なりとも影響を受けると考えられるアプリケーションとしては、1月2日を平日であると仮定するであろう給与支払プログラムのようなものがある。1900年の2日目は平日だが、2000年のその日は日曜日である。この問題によって影響を受ける別の種類のプログラムは、特殊な曜日に他のプログラムを実行するプログラムである。これらはまた、おそらく日曜日に実行し、木曜日や金曜日にシャットダウンするであろう。
もしもそのプログラムが適切にデータを修正することによって、2000年に柔軟に対処したいのであれば、この問題のいたるところに作用するルーチンを作成しなければならない。アプリケーションが曜日にアクセスするときはいつでも、ルーチンは年を判断するように作成される必要がある。もしその年が1900年代にあるならば、そのときの曜日は正確である。しかしその年が2000年以上であるならば、そのときのルーチンは2日前の曜日に移動しなければならない。もし1900年のある日が金曜日として参照されるならば、2000年のその日は水曜日に変更されなければならない。つぎに示すプログラムPROG6.CBLが、そのような技法を例示する。

```
001010 IDENTIFICATION        DIVISION.
001020 PROGRAM-ID.           PROG6.
001030 ENVIRONMENT           DIVISION.
001040 INPUT-OUTPUT          SECTION.
001050 FILE-CONTROL.
001060      SELECT  OLD-FILE  ASSIGN  TO  "A:MYFILE3.DAT".
001070      SELECT  NEW-FILE  ASSIGN  TO  "A:MYFILE4.DAT".
001080 DATA                  DIVISION.
001090 FILE                  SECTION.
```

```
001100 FD  OLD-FILE.
001110 01  OLD-DATE.
001120     05   OD-DA-YR      PIC 9(2).
001130     05   OD-DOW        PIC 9(1).
001140     05   FILLER        PIC X(2).
001150 FD  NEW-FILE.
001160 01  NW-DATE.
001170     05   NW-YR         PIC 9(3).
001180     05   NW-DOW        PIC 9(1).
001190 PROCEDURE              DIVISION.
001200 A1. OPEN   INPUT  OLD-FILE  OUTPUT     NEW-FILE.
001210 A2. READ   OLD-FILE  AT  END  GO  TO  A3.
001220     MOVE   OD-DA-YR  TO  NW-YR.
001230     IF  OD-DA-YR  IS  <  49
001240         IF  OD-DA-YR  >  1
001250               COMPUTE  NW-DOW  =  OD-DOW  -  2
001260         ELSE IF   OD-DA-YR  =  0
001270               MOVE  5  TO  NW-DOW
001280         ELSE IF   OD-DA-YR  =  1
001290               MOVE  7  TO  NW-DOW.
001300     IF  OD-DA-YR   IS  >  48
001310               MOVE  6  TO  NW-DOW
001320     END-IF.
001330     WRITE  NW-DATE  AFTER  1.
001340     DISPLAY  NW-DATE.
001350     GO  TO  A2.
001360 A3. CLOSE  OLD-FILE  NEW-FILE.
001370     STOP     RUN.
```

ここで曜日は、OD-DOW　PIC　9(1). のなかに数字の１～７までが入力される。各曜日の数字は以下のように指定される。　さらにこの結果から、各年の曜日が把握される。

７ MONDAY 、 ６ SUNDAY 、 ５ SATURDAY 、 ４ FRIDAY

３ THURSDAY 、２ WEDNESDAY 、１ TUESDAY

1900 年１月１日は、月曜日

2000 年１月１日は、土曜日

2001年1月1日は、月曜日
2002年1月1日は、土曜日
2049年1月1日は、日曜日

日付に関する年や曜日以外に、別の側面として日にち計算がある。多くのアプリケーションでは、二つの日にちの間に、何日経過しているかを判断することが必要となる。この作業を達成するために、COBOL は異なるタイプのデータ構造を提供する。日形式（YYDDD）は、ある特定年における通算日を表す。この構造において、YY は年に関連するし、DDD はその年の何日目である日にちを示している。12月31日は通常年では、365であり、閏年では366である。こうした日付形式の利用は、その日付が同じ年である限り、日付計算を全く簡単な計算にする。つぎにその計算の例をプログラムで示そう。

```
01   TWO-DATES.
     05  FIRST-DATE.
         10   FIRST-YEAR        PIC 9(2).
         10   FIRST-DAY         PIC 9(3).
     05  SECOND-DATE.
         10   SEC-YEAR          PIC 9(2).
         10   SEC-DAY           PIC 9(3).
01   DAYS-DIFFERENCE            PIC 9(3).

     MOVE  97234    TO  FIRST-DATE.
     MOVE  97100    TO  SECOND-DATE.
     SUBTRACT  SEC-DAY  FROM  FIRST-DAY  GIVING  DAYS-DIFFERENCE.
```

上記のコードの一部において、日形式は同じ年の場合には、通日をいかに容易に提供できるかが分かる。この形式では、もしも二つの日にちが、年の境界にまたがるときの処理を決めておくべきである。日形式の YY 部分は、多くの助けは必要とはしない。
たとえば、1997年12月31日（97365）と1998年1月1日（98001）との単純な引き算から636日の差が出るであろう。年境界をまたぐ日にち計算は、その差の出し方に考慮しなければならない。つぎのコードはその問題を処理する一つの方法である。

```
MOVE  98030   TO  FIRST-DATE.
MOVE  97300   TO  SECOND-DATE.
IF   FIRST-DATE  >   SECOND-DATE
      ADD   365   TO  FIRST-DAY
```

```
END-IF.
SUBTRACT SEC-DAY FROM FIRST-DAY GIVING DAY-DIFFERENCE.
```

このコードは古い年の継続として新しい年における日にちを処理する。98030はあたかも実際には97395であるかのように扱う。この技法はプログラムによって、95日の差が計算できるようにしてある。また千年紀が変化するときはこのコードでは、FIRST-YEARが00であり、SEC-YEARが99であれば、そのIF条件が偽であり、そしてDAYS-DIFFERENCEは95ではなく、270になるであろう。幸運にも1989年拡張COBOLは、日付計算をするための良い方法を提供している。つまりそこにはつぎの多くの固有関数が用意されている。それは他の言語や表計算言語にも役立つ標準的なタイプの操作を行うものである。これらの関数を使って、正しい日にち計算をしたり、2000年問題を正確に実行できるための方法を見ていくことにしよう。

もしも電子計算機の表計算言語を用いて仕事をしていると、つぎのことに気付くであろう。表計算の場合、日付はYYYYMMDD形式を採っていないことである。むしろ表計算は、ある任意の開始時点から日数のカウントをするのである。たとえば、EXCELでは、1900年1月1日を開始時点としている。言い換えれば、EXCELは1900年1月1日を第1日として処理する。1900年1月1日からのそれぞれの日は、連続して番号付けられた整数である。日付計算は、単にこれらの整数に加算されたり、またはそこから減算される事柄である。定義づけられた開始日前のいかなる日付も無効である。

COBOLの固有関数は、プログラマが表計算におけると同じような種類の整数日付関数を用いることができるようになっている。しかしながら、COBOLについての任意の開始時点は、1601年の1月1日である。この初期の日付は、EXCELが処理できるよりももっと多くの日付の思考を許すものであり、大抵の日付アプリケーションにとって十分なものであるはずである。他方、初期の開始日付は、現在の日付の整数価値がかなり大きいものであることを意味する。たとえば、2000年1月1日は、145732の整数価値である。これはプログラムとして、整数日付領域に対して少なくとも6桁を用意しなければならないことを意味する。

整数形式日付は、日付計算の問題を容易にするけれども、この形式で日付を読むには不便を感じる。そのためにCOBOLは、日付を標準形式から整数に換算し戻す関数を提供している。1989年拡張COBOLは、プログラマが使用できるようにつぎに示す6つの固有の暦関数を提供する。

① **CURRENT-DATE**　（現在のシステム日付をYYYYMMDD形式に戻す）
② **WHEN-COMPILED**　（コンパイル日付をYYYYMMDD形式に戻す）
③ **DATE-OF-INTEGER**　（YYYYMMDDを整数に換算）
④ **DAY-OF-INTEGER**　（YYYYDDDを整数に換算）

⑤ **INTEGER-OF-DATE** （ 整数日付を YYYYMMDD 形式に換算 ）
⑥ **INTEGER-OF-DAY** （ 整数日付を YYYYDDD 形式に換算 ）

最初の二つの関数は、特定の価値を戻すためにアーギュメント(引数)を必要とはしない。残りの四つの関数は、COBOL に日付を整数に換算したり、またその逆のことをさせるルーチンである。これらの関数は同じように作用する。整数日付関数がどのように作用するかを検討することによって、残りの関数の使用方法を容易に知ることができよう。その具体例として、つぎに示すのが、PROGA2.CBL である。

```
001010 IDENTIFICATION          DIVISION.
001020 PROGRAM-ID.             PROGA2.
001030 ENVIRONMENT             DIVISION.
001040 CONFIGURATION           SECTION.
001050 SOURCE-COMPUTER.        FLORA.
001060 OBJECT-COMPUTER.        FLORA.
001070 SPECIAL-NAMES.          CONSOLE  IS  CRT.
001080 DATA                    DIVISION.
001090 WORKING-STORAGE         SECTION.
001100 01  DATE-DATA.
001110     05  DATE-INTEGER   PIC 9(06).
001120     05  DATE-INPUT.
001130         10  DATE-YEAR  PIC 9(04).
001140         10  DATE-MONTH  PIC 9(02).
001150         10  DATE-DAY   PIC 9(02).
001160     05  DATE-RDF  REDEFINES  DATE-INPUT  PIC 9(08).
001170     05  LAST-DATE-SW   PIC X.
001180         88   LAST-DATE  VALUE  "N" "n".
001190 PROCEDURE              DIVISION.
001200 CONVERT-DATE.
001210     MOVE  "Y"  TO  LAST-DATE-SW.
001220     PERFORM   GET-DATE.
001230     PERFORM   UNTIL  LAST-DATE
001240       COMPUTE  DATE-INTEGER  =  FUNCTION
001250                                 INTEGER-OF-DATE (DATE-RDF)
001260       DISPLAY  DATE-INTEGER   AT  0801
001270       DISPLAY  "DO  YOU  WISH TO  CONTINUE (Y OR N)?"  AT  0901
```

```
001280        ACCEPT  LAST-DATE-SW  AT  1001
001290        IF  NOT  LAST-DATE
001300            PERFORM  GET-DATE
001310        END-IF
001320      END-PERFORM
001330      STOP   RUN.
001340 GET-DATE.
001350        DISPLAY  "ENTER  YEAR  IN  -YYYY-  FORMAT  "  AT  0101
001360        ACCEPT  DATE-YEAR      AT  0201
001370        DISPLAY  "ENTER  MONTH IN  -MM-    FORMAT  "  AT  0301
001380        ACCEPT  DATE-MONTH     AT  0401
001390        DISPLAY  "ENTER  DAY   IN  -DD-    FORMAT  "  AT  0501
001400        ACCEPT  DATE-DAY       AT  0601.
```

PROGA2.CBL で示したプログラムは、日付整数を使用している。そのプログラムでは、ユーザから年月日を受け取り、日付の整数価値に戻すものである。たとえば、誕生日として、1900 年 2 月 29 日、2000 年 2 月 29 日、あるいは好きな日付を入力して、プログラム・テストを実行してみることもできる。プログラムは、無効な日付には 0 を戻すであろう。このプログラムから、いくつかのポイントを指摘しよう。6 行目の整数領域は 6 桁の長さをもつ。上述したように 6 桁は、現在日付の整数価値を格納するのに必要である。11 行目では、DATE-RDF は、DATE-INPUT を再定義している。その理由は、関数日付整数は、入力パラメータとして基本項目を要求するからである。21 行目では、「関数」というキーワードは、日付整数が固有の関数であって、識別子ではないことを COBOL に宣言するものである。その構文は、 **FUNCTION　INTEGER-OF-DATE　（引数）**
その引数は、YYYYMMDD 形式の基本項目であり、その関数は 6 桁数字を戻す。

固有関数を用いる日付計算は、つぎの例を見ても、全く簡単なものと言えよう。

```
COMPUTE  NO-OF-DAYS  =  FUNCTION  INTEGER-OF-DATE  (DATE-1)
                     -  FUNCTION  INTEGER-OF-DATE  (DATE-2)
```

この命令文は、DATE-1 と DATE-2 を整数に換算し、DATE-1 から DATE-2 を差し引きその残りを NO-OF-DAYS に記録するものである。このように固有の関数を用いるための一つの利点は、閏年問題を避けることができ、あるいはまた最少化される点にあろう。

さて、ここで問題となった閏年について考えてみよう。2000年に柔軟でないプログラムは、時々閏年問題に関連している別のプログラムによって影響をうける。ここにそのジレンマがある。今のところ多くのアプリケーションは19世紀を仮定している。それゆえ、2000年は、アプリケーションが全ての年を表すために、00年を使用するけれども、それが1900年であると仮定される。しかしながら、1900年と異なり、2000年は閏年である。これは2月に1日が加算されることを意味する。1900年の2月の最後の日は、2月28日だが、2000年の場合は2月29日となる。閏年は,16世紀に作られた、ズレをもたらしたグレゴリア暦における主意を説明するためのものであった。その主意は、太陽暦（365日5時間48分46秒）とグレゴリア暦（ 1601年1月1日（月曜日）を第1日として計算をする通算日数を求めるものである。その通日を7で割ってその余りが1であれば月曜日、2であれば火曜日とする。いわゆるグレゴリア通日を用いるもの ）との約11分の違いによって引き起こされるものである。

閏年は4年ごとに発生する。そしてそのいずれもが等しく4で割り切れる年である。しかしながら、00に終わる年はその例外である。それらはどれも400で割り切れなければ、閏年ではない。アプリケーションでは、日付が2000年を反映するように修正されないと、閏年のために訂正するルーチンを含めなければならない。3月1日が関連するたびに、アプリケーションは、当年は2000年であるのか、そして前日が2月28日なのか29日なのかを見極めなければならない。そのために必要な修正例は、つぎのプログラムのPROG7.CBLで示すことにしよう。

```
001010 IDENTIFICATION        DIVISION.
001020 PROGRAM-ID.           PROG7.
001030 ENVIRONMENT           DIVISION.
001040 INPUT-OUTPUT          SECTION.
001050 FILE-CONTROL.
001060     SELECT  OLD-FILE  ASSIGN  TO  "A:MYFILE5.DAT".
001070     SELECT  NEW-FILE  ASSIGN  TO  "A:MYFILE6.DAT".
001080 DATA                  DIVISION.
001090 FILE                  SECTION.
001100 FD  OLD-FILE.
001110 01  OLD-DATE.
001120     05   OD-YR        PIC 9(2).
001130     05   OD-MO        PIC 9(2).
001140     05   OD-DA        PIC 9(2).
001150     05   FILLER       PIC X(2).
001160 FD  NEW-FILE.
```

```
001170 01  NW-DATE.
001180     05   NW-YR          PIC 9(2).
001190     05   NW-MO          PIC 9(2).
001200     05   NW-DA          PIC 9(2).
001210 PROCEDURE               DIVISION.
001220 A1. OPEN   INPUT  OLD-FILE  OUTPUT     NEW-FILE.
001230 A2. READ   OLD-FILE  AT  END  GO  TO  A3.
001240     MOVE   OD-YR  TO  NW-YR.
001250     IF  OD-YR  IS  =  00
001260         IF  OD-MO  =  03  AND  OD-DA  =  01
001270             MOVE  02  TO  NW-MO
001280             MOVE  29  TO  NW-DA
001290*          END-IF
001300     END-IF.
001310     WRITE  NW-DATE  AFTER  1.
001320     DISPLAY  NW-DATE.
001330     GO  TO  A2.
001340 A3. CLOSE  OLD-FILE  NEW-FILE.
001350     STOP     RUN.
```

一般的に、閏年を決める法則が混乱している。大抵の人が知っているのは、閏年はその年の価値がいずれも４で割り切れるときに発生する、というものである。多くの人が知らないのは、00年で終わる年は閏年ではない、ということである。本当に物事をもっと難しくするのは、400年毎の２月29日は、00で終わる年に発生することである。だから1600年、2000年、2400年は閏年である。このようにCOBOLでは、固有関数の日付整数によって、閏年が適切に管理され、それが無効な日付を見つけるときに、0の価値を戻すであろう。それゆえ関数は、また日付を有効にするための方法を提供する。0の価値について検査することによって、そのプログラムは無効な日付を見つけるたびに、その問題の箇所を見つけることができよう。適切なルーチンが、問題処理のために作成されるはずである。

つぎに示すプログラム、PROGB.CBLは2000年問題をクリアし、閏年を適切に処理する退職プログラムである。このプログラムは1989年COBOLの固有関数を活用し、また日付換算ユーティリティプログラムのPROGC.CBLを呼び出すものである。このユーティリティ・プログラムは、COBOL 2000の固有関数、YEAR-TO-YYYYを真似ている。

前述したように、現在日付固有関数はYYYYMMDD形式でシステム日付を戻し、ACCEPT　～　FROM　DATE文で、取り入れることができる。99行目では、システム

日付を得るために現在日付を用いる。現在日付についてのもっと多くのものが、COBOL 2000 上の付録で見つけることができる。100 行目は注釈行である。この行はシステム日付を 2000 年 1 月 1 日に設定をする。7 カラム目の＊を削除し、再コンパイルすることによって、2000 年にそのプログラムがどのように動くかを知るために、プログラムをテストすることができる。そのユーティリティ・プログラムの 24 行目を必ず変更してから再コンパイルしてみるとよい。

112 行目において、そのプログラムは 50 行目に始まるデータ集団項目である。DATE-CONVERSION-DATA を用いている、ユーティリティ・プログラムを呼び出す。そのユーティリティ・プログラムは最初の引数である、「CNV-YY」の価値を 2 桁年でとる。そして「CNV-YEAR」に 4 桁年で戻す。(同様の CALLS は、117. 122 行目に発生する。) なぜならば、そのプログラムは適切に世紀を決める必要がある。すなわち「ウィンドウ」が必要である。そのウィンドウは 100 年の幅である。「CNV-WINDOW」領域は、その幅の最高年を明記するのに役立つ。PROGC.CBL プログラムは、プログラムが戻すことができる最近の年を決定するのに、CNV-WINDOW を現在年に加算する。そのようにもし、現在年が 2000 年であり、CNV-WINDOW が 15 であるならば、1916 年から 2015 年まで 4 桁年を戻すことができる。2000 年の現在年や WINDOWS-MAX は、−15 年まで設定し、ユーティリティは、1886 から 1985 までの 4 桁年を戻す。

もし年が、過去から同様に未来になるならば、ウィンドウ価値は 50 で設定されるべきである。もし年の全てが現在年よりも小さいものであると思われるならば、ウィンドウ価値は、0 であるべきである。

PROGB.CBL において、127 行目とそのつぎの行は、2 月 29 日に生れた従業員に対する退職日付問題を扱う。この日付に生れた従業員は、65 年後の 2 月 29 日には退職しないであろう。そのプログラムは固有関数「INTEGER-OF-DATES'S」（日付整数）を、日付を有効にするために利用する。もしもその関数が 0 を戻すなら、提案した日付は無効であり、プログラムはその日付を 3 月 1 日に変更する。この技法は、単にもしソース・データ（すなわち誕生日）が以前に有効にされていたならば、適切となる。

```
001010 IDENTIFICATION          DIVISION.
001020 PROGRAM-ID.             PROGB.
001030 ENVIRONMENT             DIVISION.
001040 INPUT-OUTPUT            SECTION.
001050 FILE-CONTROL.
001060     SELECT  EMPLOYEE-FILE  ASSIGN  TO  "A:A.DAT".
001070     SELECT  OUT-FILE       ASSIGN  TO  "A:BB.DAT".
001080 DATA                    DIVISION.
001090 FILE                    SECTION.
```

```
001100 FD  EMPLOYEE-FILE.
001110 01  EMPLOYEE-RECORD.
001120     02  AAA              PIC X(34).
001130     02                   PIC X(02).
001140 FD  OUT-FILE.
001150 01  OUT-REC.
001160     02                   PIC X(80).
001170 WORKING-STORAGE           SECTION.
001180 01 EMPLOYEE-DATA.
001190     05  EMP-NUM          PIC X(05).
001200     05  EMP-NAME.
001210         10  EMP-LAST     PIC X(15).
001220         10  EMP-INIT     PIC X(02).
001230     05  EMP-BIRTHDATE.
001240         10  EMP-BIRTH-YR  PIC 9(02).
001250         10  EMP-BIRTH-MO  PIC 9(02).
001260         10  EMP-BIRTH-DA  PIC 9(02).
001270     05  EMP-SERVICE-DATE.
001280         10  EMP-SERVICE-YR PIC 9(02).
001290         10  EMP-SERVICE-MO PIC 9(02).
001300         10  EMP-SERVICE-DA PIC 9(02).
001310 01  DATA-REMAINS-SW     PIC X(02).
001320     88  NO-DATA-REMAINS   VALUE  "NO".
001330 01  INDIVIDUAL-FIELDS.
001340     05  IND-AGE          PIC 9(02).
001350     05  IND-SERV-YEARS PIC 9(02).
001360     05  IND-RET-DATE.
001370         10  IND-RET-YR PIC 9(04).
001380         10  IND-RET-MO PIC 9(02).
001390         10  IND-RET-DA PIC 9(02).
001400     05  IND-RET-DATE-RDF  REDEFINES
001410             IND-RET-DATE  PIC 9(08).
001420     05  IND-RET-INT-DATE  PIC 9(06).
001430 01  TODAYS-DATE.
001440     05  TODAYS-YR        PIC 9(04).
001450     05  TODAYS-MO        PIC 9(02).
```

```
001460     05  TODAYS-DA       PIC 9(02).
001470 01  CONSTANTS.
001480     05  RETIRE-AGE      PIC 9(02)  VALUE  65.
001490     05  YEAR-TO-YYYY    PIC X(07)  VALUE "PROGC".
001500 01  DATE-CONVERSION-DATA.
001510     05  CNV-YY          PIC 9(02).
001520     05  CNV-WINDOW      PIC S9(02)  VALUE ZERO.
001530     05  CNV-YEAR        PIC 9(04).
001540 01  HEADING-LINE-1.
001550     05                  PIC X(05)  VALUE  SPACES.
001560     05                  PIC X(10)  VALUE  "EMPLOYEE".
001570     05                  PIC X(17)  VALUE  SPACES.
001580     05                  PIC X(09)  VALUE  "SERVICE".
001590     05                  PIC X(10)  VALUE  "RETIREMENT".
001600     05                  PIC X(29)  VALUE  SPACES.
001610 01  HEADING-LINE-2.
001620     05                  PIC X(07)  VALUE  SPACES.
001630     05                  PIC X(14)  VALUE  "NAME".
001640     05                  PIC X(07)  VALUE  "INIT".
001650     05                  PIC X(06)  VALUE  "AGE".
001660     05                  PIC X(08)  VALUE  "YEARS".
001670     05                  PIC X(10)  VALUE  "DATE".
001680     05                  PIC X(28)  VALUE  SPACES.
001690 01  DETAIL-LINE.
001700     05                  PIC X(05)  VALUE  SPACES.
001710     05  DET-LAST        PIC X(15).
001720     05                  PIC X(02)  VALUE  SPACES.
001730     05  DET-INIT        PIC X(02).
001740     05                  PIC X(05)  VALUE  SPACES.
001750     05  DET-AGE         PIC 9(02).
001760     05                  PIC X(05)  VALUE  SPACES.
001770     05  DET-SERV-YEARS  PIC 9(02).
001780     05                  PIC X(02)  VALUE  SPACES.
001790     05  DET-RET-DATE.
001800         10  DET-RET-MO  PIC Z9.
001810         10               PIC X      VALUE  "/".
```

```
001820         10  DET-RET-DA PIC Z9.
001830         10             PIC X      VALUE  "/".
001840         10  DET-RET-YR PIC 9(04).
001850 PROCEDURE              DIVISION.
001860 100-PREPARE-RETIREMENT-REPORT.
001870     OPEN  INPUT  EMPLOYEE-FILE  OUTPUT  OUT-FILE.
001880     PERFORM   210-GET-TODAYS-DATE
001890     PERFORM   230-WRITE-HEADERS
001900     PERFORM   UNTIL  NO-DATA-REMAINS
001910        READ   EMPLOYEE-FILE  INTO  EMPLOYEE-DATA
001920           AT  END      SET  NO-DATA-REMAINS  TO  TRUE
001930           NOT  AT  END  PERFORM  260-PROCESS-DETAIL
001940     END-READ
001950     END-PERFORM
001960     CLOSE     EMPLOYEE-FILE  OUT-FILE
001970     STOP      RUN.
001980 210-GET-TODAYS-DATE.
001990     MOVE  FUNCTION  CURRENT-DATE  TO  TODAYS-DATE.
002000*    MOVE  "20000101"     TO   TODAYS-DATE.
002010 230-WRITE-HEADERS.
002020     WRITE  OUT-REC   FROM  HEADING-LINE-1   AFTER  1
002030     WRITE  OUT-REC   FROM  HEADING-LINE-2   AFTER  1
002040     INITIALIZE  OUT-REC.
002050 260-PROCESS-DETAIL.
002060     PERFORM  310-CALCULATE-EMP-AGE
002070     PERFORM  330-CALCULATE-EMP-SERVICE
002080     PERFORM  360-CALCULATE-IND-RET-DATE
002090     PERFORM  390-WRITE-DETAIL-LINE.
002100 310-CALCULATE-EMP-AGE.
002110     MOVE     EMP-BIRTH-YR  TO  CNV-YY
002120     CALL     YEAR-TO-YYYY  USING  DATE-CONVERSION-DATA
002130     COMPUTE  IND-AGE  =  TODAYS-YR  -  CNV-YEAR
002140                       +  (TODAYS-MO -  EMP-BIRTH-MO)  /  12.
002150 330-CALCULATE-EMP-SERVICE.
002160     MOVE     EMP-SERVICE-YR  TO  CNV-YY
002170     CALL     YEAR-TO-YYYY  USING  DATE-CONVERSION-DATA
```

```
002180     COMPUTE  IND-SERV-YEARS  =  TODAYS-YR  -  CNV-YEAR
002190                        +  (TODAYS-MO -  EMP-SERVICE-MO)  /  12.
002200 360-CALCULATE-IND-RET-DATE.
002210     MOVE  EMP-BIRTH-YR  TO   CNV-YY
002220     CALL  YEAR-TO-YYYY  USING  DATE-CONVERSION-DATA
002230     ADD   RETIRE-AGE    TO   CNV-YEAR  GIVING  IND-RET-YR
002240     MOVE  EMP-BIRTH-MO  TO   IND-RET-MO
002250     MOVE  EMP-BIRTH-DA  TO   IND-RET-DA.
002260*  TEST  FOR  INVALID  FEBRUARY  29  RETIREMENT  DATE
002270     COMPUTE  IND-RET-INT-DATE  =  FUNCTION  INTEGER-OF-DATE
002280                                             (IND-RET-DATE-RDF)
001290*  WHEN  FOUND, SET  DATE  TO  MARCH  1.
002300     IF  IND-RET-IN-DATE  =  ZERO
002310           ADD  1  TO  IND-RET-MO
002320           MOVE 1  TO  IND-RET-DA
002330     END-IF.
002340 390-WRITE-DETAIL-LINE.
002350     MOVE  EMP-LAST      TO   DET-LAST
002360     MOVE  EMP-INIT      TO   DET-INIT
002370     MOVE  IND-AGE       TO   DET-AGE
002380     MOVE  IND-SERV-YEARS  TO  DET-SERV-YEARS
002390     MOVE  IND-RET-MO    TO   DET-RET-MO
002400     MOVE  IND-RET-DA    TO   DET-RET-DA
002410     MOVE  IND-RET-YR    TO   DET-RET-YR
002420     WRITE  OUT-REC      FROM  DETAIL-LINE  AFTER  1.


001010 IDENTIFICATION        DIVISION.
001020 PROGRAM-ID.           PROGC.
001030 ENVIRONMENT           DIVISION.
001040 CONFIGURATION         SECTION.
001050 SOURCE-COMPUTER.      FLORA.
001060 OBJECT-COMPUTER.      FLORA.
001070 SPECIAL-NAMES.        CONSOLE  IS  CRT.
001080 DATA                  DIVISION.
001090 WORKING-STORAGE       SECTION.
```

```
001100 01  TEMPORARY-DATA.
001110     05  WORK-YEAR.
001120         10  WORK-HIGH-YY  PIC 9(02).
001130         10  WORK-LOW-YY   PIC 9(02).
001140     05  WORK-YYYY   REDEFINES  WORK-YEAR  PIC 9(4).
001150 LINKAGE                SECTION.
001160 01  LS-CONVERSION-DATA.
001170     05  LS-YY           PIC 9(02).
001180     05  LS-WIND         PIC S9(02).
001190     05  LS-YYYY.
001200         10  LS-HIGH-YY PIC 9(02).
001210         10  LS-LOW-YY  PIC 9(02).
001220 PROCEDURE              DIVISION  USING  LS-CONVERSION-DATA.
001230     MOVE  FUNCTION    CURRENT-DATE  TO  WORK-YEAR.
001240*    MOVE  2000   TO   WORK-YYYY.
001250     ADD   LS-WIND     TO   WORK-YYYY.
001260     MOVE  WORK-HIGH-YY  TO   LS-HIGH-YY.
001270     MOVE  LS-YY  TO   LS-LOW-YY.
001280     IF  LS-YY  >  WORK-LOW-YY
001290         SUBTRACT  1   FROM  LS-HIGH-YY
001300     END-IF.
001310     EXIT   PROGRAM.
```

５．その他の対処法と2000年ツール

日付を計算するためのその他の方法として、「日付圧縮」方法を示すことにしよう。この方法は、追加的なディスク容量を要求しないものであるが、かなり広い範囲の年を処理することができる。しかしながら、いわゆる理論的には優れてはいるが、実行速度が遅くなったり、他の会社とデータ共有していれば問題を引き起こすものである。

この「日付圧縮」は、４桁年価値のデータ形式を修正することから始める。その結果それらは２桁領域で表される。数字もしくは英字領域で４桁年を記憶する代わりに、それらは異なる形式で２桁文字として記憶される。もしこの日付圧縮法を用いようとするならば、圧縮形式と実際の日付との間の日付を換算する、単なる一組のプログラムが必要となる。既存のプログラムからこのプログラムを呼び出し、そこで入出力文が実行されることになる。しかしながら、既存のデータを移しかえなければならない。その結果、それは圧縮形式で表される。これはつぎの場合に、問題を引き起こす。すなわち同じソリューションを

用いていない他の会社とのデータを共有する場合である。

それでは、実際に日付圧縮をする場合の形式を三つに分けて見てみよう。

① 16 進数形式

ここでは 2 桁の 16 進数が任意に選んだ基準年からの派生価値を表す。たとえばその基準年を 1900 として選べば、1999 年と 2006 年は、ここではつぎのように示す。

＊現在年（1999 年）を表す派生価値は、16 進数で 63 として定義される。

```
01    CURRENT-YEAR   PIC  X(2)  VALUE  "63".
```

＊2006 年を表す派生価値は、16 進数で 6A として定義される。

```
01    YEAR-2006          PIC  X(2)   VALUE  "6A".
```

日付圧縮によってシステムに 16 進数を用いることは、作業ができる年数は 255 年である。基準年を 1900 年とすると、プログラムは 2155 年まで操作できることを意味している。

② CYYDDD 形式

この CYYDDD 圧縮方法は、1900 年を開始時点として用い、その日付から経過した世紀、年、日をカウントする。最初の桁、C は 1900 年から経過した世紀の数であり、第 2、第 3 桁の YY は、その世紀に対する 2 桁年と同じものである、経過年数を表す。最後の 3 桁の DDD は、ユリウス日付（ 旧太陽暦 ）と同じ方法で、年の開始以来経過した日数を表す。

＊ 基準年として 1900 年から始めて、2000 年 1 月 1 日はつぎのように表される。

```
01    FIRST-DAY-YEAR2000    PIC  9(6)   VALUE  10101.
```

この圧縮方法を用いるならば、任意の基準日付から 100 年範囲内（1 世紀、99 年と 365 日）に発生する日付を操作できよう。

③DDDDDD 形式

この DDDDDD 圧縮方法は、経過した世紀、年、日の数をカウントしないということを除いて、経過した日数をまさにカウントするということは、CYYDDD 圧縮方法に類似している。たとえば、基準年を 1900 年として、この方法を用いて、2000 年 1 月 1 日の日付を圧縮したいのなら、つぎのように日付を表すことができる。

```
01    FIRST-DAY-YEAR2000    PIC  9(6)   VALUE   036526.
```

もしこの圧縮方法を用いるならば、基準年から 2700 年まで発生する日付を操作できよう。

これまでに日付圧縮方法を一瞥したが、これについては実施するためにはつぎの段階を経なければならない。COBOLプログラムにおいて、日付圧縮ソリューションを実施するための一般的な段階を示しておこう。

(1) する必要のある仕事の量を評価し、どこに変更をしなければならないのかを知るために、「調査ツール」および「衝撃分析ツール」に示されているツールを用いよ。
(2) どんなにミスのあるソースモジュールをも再検討し、作り直せ。
(3) 経営者に言って、仕様書の作成を要求し、それをプログラマに実施させよ。
この仕様書は、第1段階で行われないさまざまなコーディング状況を自ら作成せよ。たとえばそれは、用いられる日付圧縮方法を記述することや日付を容易に換算できるような追加的情報も与えるべきである。たとえば日付が、１６進数で圧縮されているならば、基準年から始まる16進数と10進数を示す表を提供すべきである。
(4) 日付圧縮方法を用いて、アプリケーションを換算せよ。
(5) データを換算せよ。
(6) アプリケーションをテストせよ。
2000年変更をもっと容易にテストするために必要なツールについての情報を集めよ。
(7) テスト段階中に行われなかったどんなミスも訂正せよ。
(8) 全てのテスト段階を終了するまで、第６．第７段階を繰り返せ。そしてプログラムでしなければならない作業を確信せよ。
(9) もしも資金的な余裕があれば、有効なプログラムであるかをテストするためのテスト・システムを構築せよ。

以上の段階は、システム・ユーザをそのプロセスに参加させるようにするには、極めて良い機会である。なぜならば、彼らはそのシステムを用い、テスト段階中に、価値あるフィードバックの提供を受けるからである。そして10進法で日付データを見られるようなユーティリティを用いたいと思うようになるかもしれない。もしもそれを可能にするユーティリティをもうすでに持っているならば、それは2000年に柔軟性があるという確信を持っても良いであろう。もしそうでなければ、テスト・ツールのいくらかがその機能を提供するであろう。

最後に、2000年ツールの特性と具体的な製品名を列挙しておこう。

2000年ツール

(1) 調査ツール　　所有しているプログラム数、その大きさや内容を評価するのに有効なツール
(2) 衝撃分析ツール　　所有しているプログラムの中で、どれだけ多くの換算作業が必要かを評価するのに有効なツール

(3) テスト・ツール　所有しているプログラムを換算した後で、それが適切かを評価するのに有効なツール
(4) 換算ツール　所有しているプログラムを実際に換算作業をするのに有効なツール
(5) 編集ツール　所有しているプログラムを2000年換算のために編集するのに有効なツール
(6) ソース修復ツール　ミスのあるソースコードを修復するのに有効なツール
(7) ツール・セット　調査、衝撃分析、コード変更のようないくらかのジョブをしたり、テストをするのに有効なツール
(8) IBM千年紀言語拡張　COBOLプログラム内のデータ項目に標識を付け、プログラムで指示した通りに、データを自動換算するツール
(9) 日付ルーチン　プログラム中で用いるのに有効な日付ルーチン

2000年ツール製品名

(1) 調査ツール　IBM　OPTI-AUDIT , Soft Audit / 2000 and Soft Audit / One , GILES , YR2K Management System , ADPAC Inventory
(2) 衝撃分析ツール　Edge Portfolio Analyser , NA 2000 SCAN , Revolve / 2000 , SEARCH 2000 , Maintenance 2000
(3) テスト・ツール　Data Ager / 400 , WITT Year 2000 , Hour Glass 2000 , TICTOC , VIA / Valid Date
(4) 換算ツール　BYPASS 2000 , INTO 2000 , Trans Century Analysis , File-Aid , Vantage Year 2000 , Hot Date 2000 / CONVERT
(5) 編集ツール　PREDITOR , ICE
(6) ソース修復ツール　Source Recovery
(7) ツール・セット　YEAR 2000.EXE , Piercom 2000
(8) IBM千年紀言語拡張　つぎのCOBOL製品に有効である。
COBOL for OS / 390&VM , COBOL for VSE , COBOL Set for AIX , ILE COBOL for AS / 400 , Visual Age COBOL (OS / 2 and Windows NT)
(9) 日付ルーチン　Trans Century Calender Routines

6. おわりに

2000年問題は、数年前から一般には知られてはいたが、ここ最近になって関心が寄せられてきた。この拙稿を纏めている1999年10月の時期から見ると、2000年1月1日はも

う今そこに差し迫ってきており、注意と警告の声がいたるところで聞かれるようになった。もうこの問題からは逃避することはできないし、またいつまでもその期限は待ってはくれない。これまでにすでにプログラムの修正や模擬テストを完了している組織体も多いが、とにかくかなり多くのプログラムとファイルを変更し、そのテストをしなければならないため、極めて困難な作業であると考えられる。情報技術部門では、日付のあらゆる発生とその日付をYYYY形式に換算しなければならないからである。

日付形式の二つの種類として、日付形式のYYYYMMDDと日にち形式のYYYYDDDのものがあったが、この二種類の日付のいずれも修正されなければならない。こうした日付修正の手助けとなったのが、COBOL85に対する1989拡張COBOLであった。そこには、COBOL固有の日付関数が用意され、2000年問題の処理対策の便宜が与えられたのである。もちろんそれらの関数を駆使し、2000年問題対処のためのプログラムを独自に開発することは望ましいことではあるが、そうした方法を採れないところでは、その他の手段として2000年ツールのソフトウェアを活用することが可能である。

これまで本稿では、世界共通の言語として汎用性のあるCOBOLプログラムによって、2000年問題の対処方法の在り方をみてきた。最新の関数をもつ、Windows NT用のCOBOLは、これまでに開発され運用されている既成プログラムの保守と、今後の新しい展開に対応するだけの能力を搭載し、これからも社会経済システムを支える重要なプログラム言語として、その役割を維持し続けるであろう。

References

Jim Keogh (1998) , COBOL Programmer's Notebook , Prentice Hall PTR , Chapter Twelve , Working to Solve the Year 2000 Problem , pp.307-329.

Robert T.Grauer , et al. (1998) , COBOL From Micro to Mainframe — Preparing for the New Millennium—, Prentice-Hall ,Chapter 19 , The Year 2000Problem , pp. 583-602.

Jon Wessler , et al. (1998) , COBOL Unleashed , Sams Publishing ,Chapter 30 , The Technical Problem and Dealing with It , pp. 833-863.